Web and Internet Protocols

In case you haven’t

- In case you haven’t already signed up for MISC here’s the link

  umisc.info/join

- Github for slides: **https://github.com/umisc/workshops/**

Overview

- OSI and TCP/IP Model
- HTTP
- HTTP Headers and Status codes
- Tools
- Practice Challenges !!!

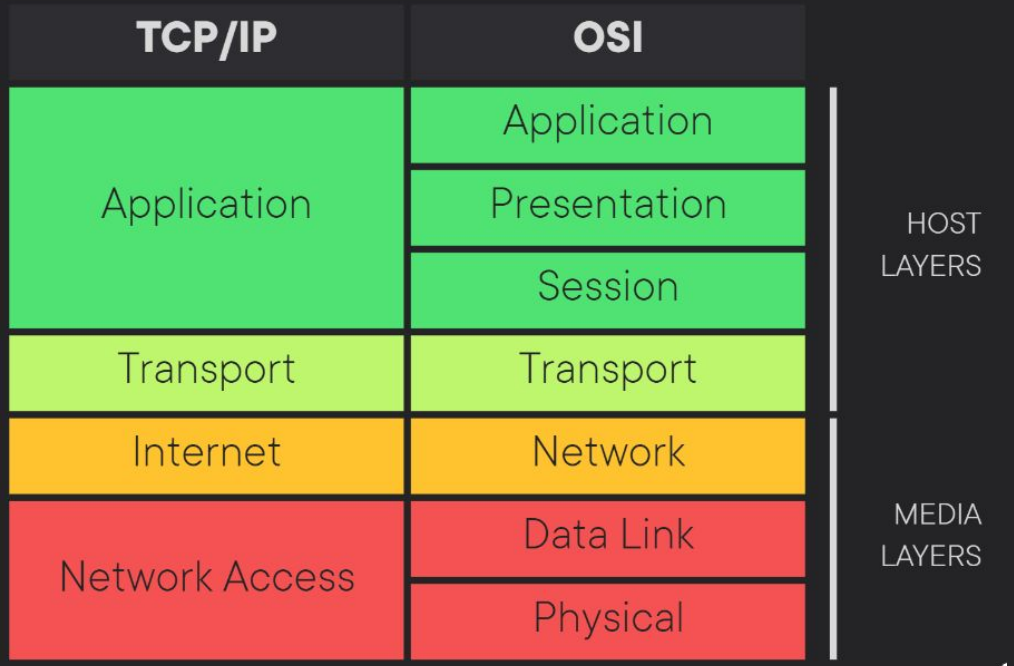
OSI Model

- The model is used as a **reference** to understand the **networks operate.**
- It consists of a **stack** with **7** layers, where each of the layers perform a specific function.
- The layers are only **interact** with their **adjacent layers**.
- It goes all the way from applications to the physical networks.

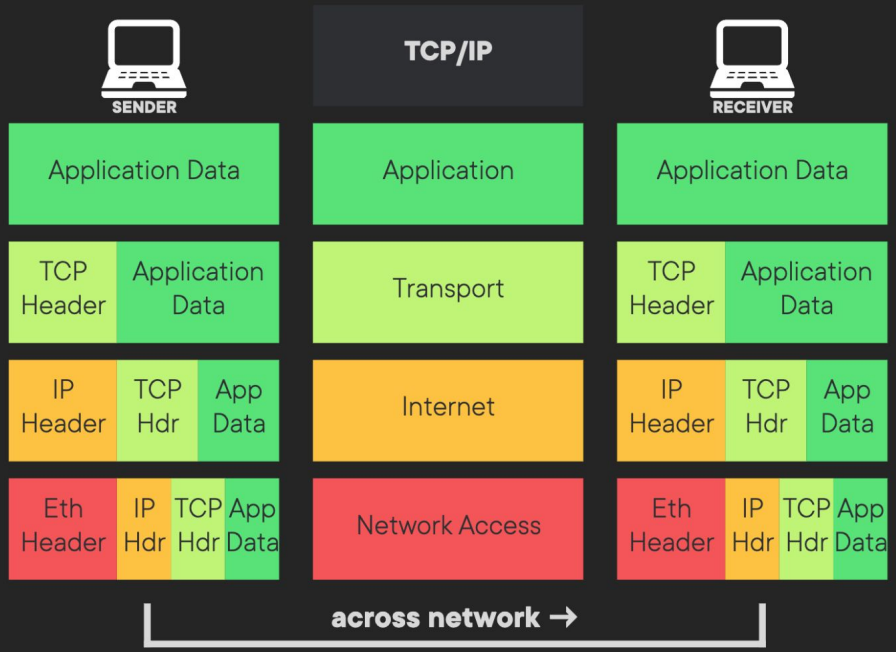
TCP/IP Model

- It is **similar** to the OSI model
- Is the **base** of how the internet works today
- Instead of 7 layers it contains **4 layers** instead which can approximately be mapped to the OSI model

TCP/IP vs OSI Model



What happens when you send a packet



Internet

- **Network of Networks** of an enormous size
- Used TCP/IP Model as a **base**
- Consists of two main parts
  - **IP Address**
  - **Port**
- You connect to services in an external network using the **public IP address** and a **port**

What is HTTP ?

- **H**yper **T**ext **T**ransfer **P**rotocol
- Communication between web servers and clients
- HTTP Requests / Response
- Loading pages , form submit and many others

HTTP is Stateless

- Every request is completely **independent**
- Each requests kind of acts like a transaction
- Programming, cookies, local storage and sessions to improve user experiences

URLS


http://www.google.com/search?q=hello
scheme
hostname
path
query

HTTP Request

HTTP message consists of

- Request Line
  - `eg: GET /search?q=hello HTTP/1.1`
  - Form `<METHOD> <PATH> <VERSION>`
- Request Header
  - eg: `Host: google.com`
  - Form: `Header: Value`
  - Can have many lines
- An empty line
- An optional message body

HTTP Response

HTTP response message consist of

- Status line
  - eg: `HTTP /1.1 200 OK`
  - form `<VERSION> <STATUSCODE> <MSG>`
- Response header
  - eg: Content-Type: text/html
  - form Header: Value
- An empty line
- An optional message body

That’s it

HTTP Request Methods

Method	Description
GET	Used to retrieve data
HEAD	Response includes only response headers
POST	Used to send data to the server
PUT	Creates/updates a resource in the server
Delete	Deletes the specified resource
Options	Requests for the capabilities of the server

Further info - https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

HTTP Request Headers

Name	Description
Connection	Controls the connection (close,keep-alive,...)
Content-Length	Length of the request body (for POST and PUT requests)
Content-Type	Type of data being sent in the request body (for POST and PUT requests)
Cookie	To maintain state
Host	Specifies the domain name of the server. Mandatory (to enable response to different pages on the same server)
Referer	Address of the previous web page
User-Agent	Identifies the user agent (client) making the request

HTTP Response Headers

Name	Description
Connection	Same as the request Connection header
Content-Length	Length of the response body
Content-Type	Type of data in the response body
Location	Used for redirects
Server	Names the server responding to the request
Set-Cookie	Instructs the client to set a cookie. The server can send multiple Set-Cookie headers to set multiple cookies

HTTP Response Codes

Code	Status Message	Description
200	OK	Standard response for successful requests
301	Moved Permanently	Redirect to the URL given in the Location header
400	Bad Request	Client side error
401	Unauthorized	Authorization required to access resource
403	Forbidden	User has insufficient permissions
404	Not Found	Unable to find resource
500	Internal Server Error	Generic error message for server side errors

Example #1

Request:

```
HEAD / HTTP/1.1
Host: www.umisc.info
```

Example #2

Request:

```
POST /login HTTP/1.1
Host: www.example.com
Content-Type: application/json
Content-Length: 48

{"username":"misccat","password":"miscpassword"}
```

Response:

```
HTTP/1.1 200 OK
Server: Express
Set-Cookie: session_id=68616820676f6f64206f6e652121
```

Example #3

Request:

```
GET /search?q=hello HTTP/1.1
Host: google.com
```

Response:

```
HTTP/1.1 200 OK
Server: gws
Content-Type: text/html
...

<!doctype html><html lang="en-AU"><head><meta charset="UTF
-8"><meta content="/images/branding/googleg/1x/googleg_sta
ndard_color_128dp.png" itemprop="image"><title>hello - Goo
gle Search</title>...
```

Cookies

- Cookies help to maintain state
- Servers instruct clients to store a cookie with the Set-Cookie response header
- Client’s responsibility to keep and send the cookie in the Cookie request header for future requests
- Cookies are used by most websites as a login feature to store sessions
    - But if the session is not saved or validated by the server side , it can be very bad

Takeaways

- HTTP requests and responses are just pieces of text formatted according to the specifications
- The server is responsible for interpreting most of the things sent in the request
- Your web browser automatically generates and sends data, but that doesn’t mean you can’t edit it yourself!

Some Useful Tools

- **nc** - create TCP and UDP connections, also has ability to listen for TCP and UDP connections. Comes installed with most Linux distributions
- **curl** - transfer data using URLs. Comes installed with most Linux distributions
- **httpie** - easy to use command line HTTP client
- **Burp Suite** - GUI program with a proxy to intercept(amongst other things) and edit HTTP traffic
- **Firefox** - web browser with good tools

NC Usage

- Connect to host example.com listening on port 443:
  - nc example.com 443
- Add headers to be sent
  - GET /login HTTP/1.1
  - Host: www.example.com
  - And many others
- Listen for TCP connections on port 1337:
  - nc -lvp 1337

Challenges!

- **Easy**
  - Web Fundamentals - https://tryhackme.com/room/webfundamentals
  - Network Fundamentals - **https://www.tryhackme.com/room/introtonetworking**
- **Intermediate/ Hard**
  - HTTP request smuggling labs in PortSwigger